

Mapping and Reducing the Brain on the Cloud

Esha Sahai and Tuhin Sahai

August 14, 2012

Abstract

The emergence of cloud computing has enabled an incredible growth in available hardware resources at very low costs. These resources are being increasingly utilized by corporations for scalable analysis of “big data” problems. In this work, we explore the possibility of using commodity hardware such as Amazon EC2 for performing large scale scientific computation. In particular, we simulate interconnected cortical neurons using MapReduce. We build and model a network of 1000 spiking cortical neurons in Hadoop, an opensource implementation of MapReduce, and present results.

1 Introduction

Cloud computing is a term that has recently received significant buzz and interest. It has been popularized by social media outlets as well as large companies that provide IT services. Though the meaning of the term “cloud computing” is context driven, typical usage refers to software or computer program execution on commodity servers at a location unknown to the user [1]. More importantly, the implementation of underlying cloud computing software (such as Hadoop) allows one to exploit distributed computation paradigms to obtain scalable solutions with fault tolerance [2]. Cloud computing has been credited with providing scalable infrastructure to Google, [3], Facebook and Twitter to name a few.

The vast trove of user generated data on the Internet has also propelled the emergence of cloud computing infrastructure [4]. Besides the immediate impact of powering Internet services, cloud computing is slowly impacting other areas of applications, whose ramifications may one day be much greater than simply organizing information on the Internet. It has converted the accessibility to large scale parallel computers from an exclusive club of rich universities and institutions to a commodity that can be purchased via the

web. For example, Amazon’s Elastic Compute Cloud (EC2 in short) allows users to rent a computer processor for as low as \$0.02 an hour [4]. Thus, for a few hundred dollars one can rent an army of computers to solve any computer problem at hand.

In this work, we investigate the possibility of using cloud services for scientific computation. Applications involving scientific computation can easily require the immense computing resources of supercomputers to extract an answer with required accuracy. For example, IBM researchers simulated a portion of the mouse brain (comprising of 22 million neurons) on a BlueGene L supercomputer with 8192 processors [5].



Figure 1: We investigate the possibility of using cloud infrastructure for scientific computation. In particular, we use a model of interconnected neurons.

As an illustration of the importance of scientific computation, Oak Ridge National Laboratory has built the Jaguar supercomputer with 299,008 processors. Parallel computers are routinely used to model the climate [6], design aircraft [7] and study nuclear fusion [8] to name a few. The available computer time for Jaguar is allocated using a competitive proposal process. Due to the utility of parallel machines, scientific institutions, universities and corporations typically invest millions of dollars annually to build or buy

compute time on supercomputers.

In this manuscript, we investigate if cloud computing is a feasible approach for conducting scientific computations. If it is, then one can simply buy computing time on the cloud as a pay-as-you-go service. This eliminates the need for maintaining large and expensive computing facilities. Note that cloud computing has previously been used to analyze the human genome [9].

2 Brief Overview of MapReduce and Hadoop

As mentioned previously, the cloud computing paradigm has enabled orders of magnitude improvement in the ability to cheaply process large amounts of data [3, 4]. Arguably, the most popular cloud computing approach that enables scalability is MapReduce [2, 3].

A MapReduce task is typically broken down into two steps: a map step which is followed by a reduce step; thus giving this parallel computing paradigm the name “MapReduce”. Note that certain applications may only require map steps or alternatively necessitate multiple reduce steps. The input data for a MapReduce job is first fed into a user specified map function. The map function processes this data and outputs multiple key-value pairs. Note that the keys in these key-value pairs need not be unique. For example, in the word count example, keys are words that may be repeated several times in a document. After the map step completes data processing, the values with the same key are combined locally at every compute node using a combine step [11]. This combine step can be viewed as a local reduce step that minimizes the amount of data transfer between nodes required during the final reduce step. The key and value list from the combine step is then input to a user-defined reduce step. The reducer performs computation on this list of keys and values to output a final key-value pair that contains the answer. In case of iterative and graph algorithms, one may need to chain a sequence of map and reduce steps [11].

Despite criticism [10], MapReduce remains a popular cloud computing paradigm that allows user-defined parallel processing of large data sets. MapReduce has been used to implement a host of different algorithms [11]. These include, word count [11], Page-rank algorithm [12], singular value decomposition [13] and clustering [13] to name a few. Once an algorithm is cast into the MapReduce paradigm [3], it becomes parallelized and ready for execution on large clusters of computers [2]. Multiple map jobs are executed in parallel to enable parallelization of input data processing. Similarly, independent reduce jobs are also executed in parallel, enabling scalability of

data processing.

The most popular MapReduce implementation is Hadoop [2]. Hadoop includes the Hadoop Distributed File System (HDFS) that manages data. It also automatically schedules jobs, restarts processes in case of machine failures, and handles data transfer between computers. Hence, once an algorithm is cast in the high-level MapReduce setting, Hadoop handles the details of the implementation of the parallel computation.

In the Hadoop system [2], NameNode was, until recently, a single point of failure. This critical compute node performs data and job assignments for nodes in the cluster and tracks job failure. This drawback has since been overcome using HA NameNode.

There are several benefits of MapReduce. The primary benefit is that MapReduce has been implemented in multiple open-source projects including Hadoop. This provides the user significant support with several examples available for demonstration and testing. We thus choose Hadoop based MapReduce to implement our neural models on the cloud.

3 Neural Models

Computational neuroscientists routinely use computer models to investigate brain dynamics [14, 15]. In particular, they have used computational models to explain various phenomena such as infant visual foraging [16], synchrony [17] and movement [18].

Brain models are typically constructed by interconnecting models of individual neurons [5, 19]. The Hodgkin-Huxley equations are an extremely popular model for describing the behavior of individual neurons [20]. In this work, we instead use the simple spiking model of a neuron developed in [21]. This model retains the accuracy of the Hodgkin-Huxley model [20], without making unrealistic simplifications routinely made in integrate-and-fire models. These simple spiking models are derived by reducing Hodgkin-Huxley equations using bifurcation methodologies [21]. Thus, this model can accurately recreate the regular spiking, fast spiking and intrinsically bursting firing patterns displayed by neurons in a rat’s motor cortex. Note that the equations to model the neuron in the simple spiking setting may be different if only one neuron is to be simulated [21].

As motivated above, to simulate individual neurons, we use the model

developed in [21],

$$\dot{v} = 0.04v^2 + 5v + 140 - u + I, \quad (1)$$

$$\dot{u} = a(bv - u), \quad (2)$$

with after spike resetting,

$$\text{if } v \geq 30 \text{ then } \begin{cases} v \rightarrow c \\ u \rightarrow u + d. \end{cases} \quad (3)$$

Here v is the membrane potential of an individual neuron, u is the membrane recovery variable that provides negative feedback to v [21] as is evident from Eqn. 1. In Eqn. 2, a (typical value of $a = 0.02$) sets the time scale of the recovery of u and b quantifies the coupling between u and v (typical value $b = 0.2$). Greater the value of b the stronger the coupling between the variables. The synaptic currents are represented through variable I .

The neuron reset rule is quantified in Eqn. 3. In particular, when $v \geq 30$, v is reset to a value of c (typical value $c = -65$ mV). Parameter d in Eqn. 3 quantifies the after spike reset value of u (typical value $d = 2$). For a more detailed explanation of the model and parameters, we point the reader to [21]. Note that, we use the above model for demonstration purposes. One could easily replace this neural model by more accurate ones such as [20].

Just as in [21], we achieve heterogeneity by assuming randomness in the parameters for the excitatory and inhibitory neurons in the network. We assume the same randomness as [21]. For excitatory neurons, we assume, $(a_i, b_i) = (0.02, 0.2)$ and $(c_i, d_i) = (-65.0, 8.0) + (15.0, -6.0)r_i^2$, where r_i is a random variable picked from the uniform distribution on $[0, 1]$. For inhibitory neurons, we assume, $(a_i, b_i) = (0.02, 0.25) + (0.08, -0.05)r_i$ and $(c_i, d_i) = (-65.0, 2.0)$. Further, the synaptic connections are represented by the S matrix. When neuron j fires, the value of v_i instantaneously changes by the entry s_{ij} [21]. We now simulate 800 excitatory and 200 inhibitory neurons that are interconnected. To implement this simulation on the cloud, we first construct a MapReduce algorithm to evolve this network of neurons.

4 MapReduce Implementation

To construct a MapReduce approach for the neural modeling of the brain on the cloud, we break the simulation task into the previously described Map and Reduce steps. Each MapReduce step evolves the network of neurons by

1 ms. To simulate the network for longer time periods, multiple MapReduce steps can be chained together as shown in Fig. 2. The output of each step is written to HDFS and is read back for the next iteration. Note that, in future work, we are looking to replace the high latency HDFS writing with solutions from the Apache Oozie and Apache Giraph projects.

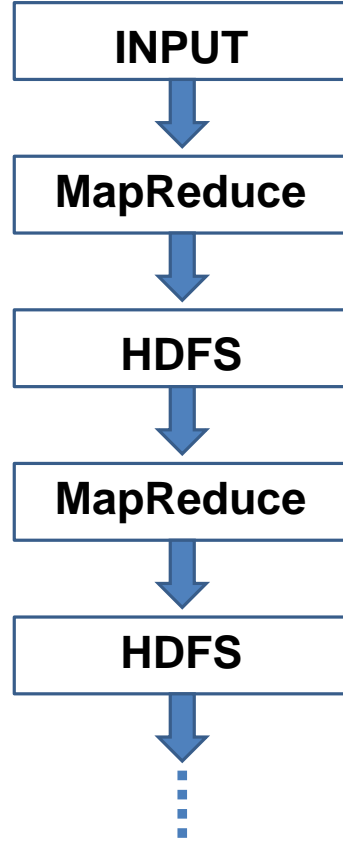


Figure 2: Our approach to chaining Hadoop jobs. This will be replaced by other Hadoop job chaining solutions in subsequent implementations.

The approach for a single MapReduce step is illustrated in algorithms 1 and 2. In the map step, the data for each neuron is first read into a data structure for each neuron. Here the key is the neuron number and the data structure is the corresponding value. Using randomly generated synaptic current values $N.I$, the membrane potential $N.v$ and membrane recovery

variable $N.u$ are updated for the current iteration. If the membrane potential $N.v$ exceeds the threshold of 30 mV, the membrane potential and recovery variables are reset according to Eqn. 3. If the n -th neuron has fired, it emits $N.S(j)$ to the j -th neuron.

Algorithm 1: Mapper for brain simulation on the cloud. The map step for each neuron emits the entire data neuronal data such as S , v , u , I along with the voltage reset $N.S(m)$ that contains firing information.

```

class MAPPER
  method MAP(Neuron id n, Neuron Data N)
     $N = \text{ReadData}()$ ; // Read the data
    if  $N.TYPE == 1$  then           // If Neuron is Excitatory
       $N.I = 5 \times \text{Random}([0, 1])$ 
    else                             // If Neuron is Inhibitory
       $N.I = 2 \times \text{Random}([0, 1])$ 
    end
     $N.I = N.I + N.Sum$ 
     $N.v = N.v + 0.5(0.04N.v^2 + 5N.v + 140 - N.u + N.I)$ 
     $N.v = N.v + 0.5(0.04N.v^2 + 5N.v + 140 - N.u + N.I)$ 
     $N.u = N.u + N.a(N.bv - u)$ 
     $N.Sum \leftarrow 0$ 
     $N.iter \leftarrow N.iter + 1$ 
    if  $N.v \geq 30$  then           // Check if Neuron has fired
      for all the Neuron id m  $\in N.S > 0$  do
         $\text{EMIT}(\text{Neuron id } m, \text{Reset data } N.S(m))$  // Emit data
      end
       $N.v \leftarrow N.c$ 
       $N.u \leftarrow N.u + N.d$ 
    end
     $\text{EMIT}(\text{Neuron id } n, \text{Neuron Data } N)$  // Pass Neuron data

```

The second step in the process is the reduce step. The reducers collect the outputs of all the neurons and compute the sum of the input synaptic current $N.Sum$ for each neuron. Note that $N.S$ contains the connections of the neuron number n to the rest of the neurons in the network [21]. The algorithmic approach for simulating the network of interconnected neurons in MapReduce is shown in Fig. 3.

Algorithm 2: Reducer for the brain simulation on the cloud. The Reducer step sums the value for $N.S$ over all the NodeData.

```

class REDUCER
  method REDUCE(Neuron id m, Neuron Data List [ $p_1, p_2, \dots$ ])
     $M \leftarrow \Phi$ 
    forall the  $p \in [p_1, p_2, \dots]$  do
      if  $IsNode(p)$  then                                // Recover Data Structure
         $M \leftarrow p$ 
      else                                                // Get firing data
         $p.Sum \leftarrow p.Sum + p$ 
      end
    end
  end

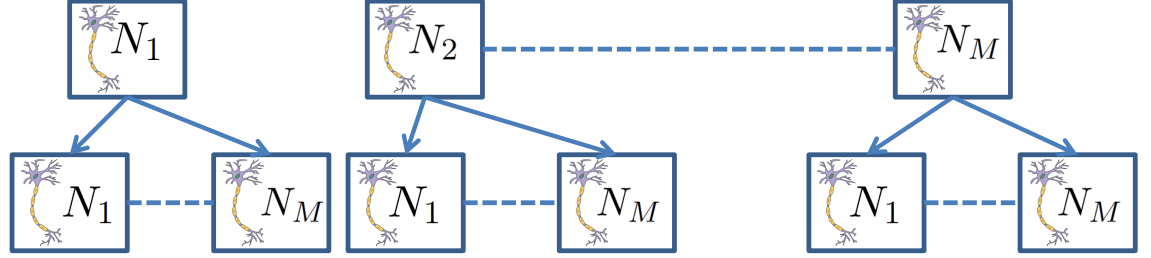
```

5 Results

We now discuss the results from running the Hadoop code for the network of 1000 neurons. We run the code for 500 ms and obtain statistically identical results as in [21]. The model is able to capture the synchronization of the neurons. In particular, Fig. 4 shows the synchronous firing of the neurons in the alpha frequency range (10 Hz). Typical (using neuron number 1 as an example) evolution of the membrane voltage v and recovery variable u is shown in Figs. 5 and 6 respectively.

The computation was done on a single processor machine with a virtualized cloud (single node cluster) for testing. As expected, the computation time was slower (by a factor of 5) than the regular code since both the size of the problem and the computing resources were significantly smaller than what would make scalability evident. However, for larger problems (millions of neurons) on the cloud, the approach is expected to provide significant speedup over monolithic implementations. This work demonstrates the feasibility of using the cloud infrastructure for scientific computing. Note, however, we do not claim that the results on the cloud will be comparable to high performance clusters for scientific computing [22]. However, we do envision that the costs will be significantly lower if one were to use commodity hardware. Moreover, these costs are expected to reduce over time as cloud computing is embraced and becomes more mainstream.

Map



Reduce

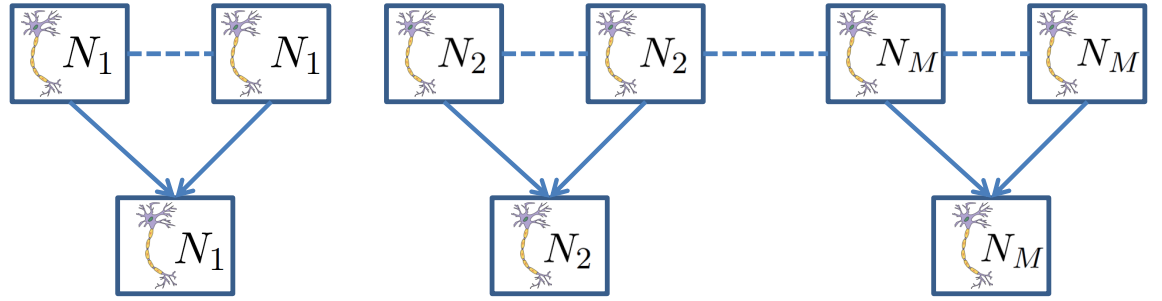


Figure 3: MapReduce algorithm for simulating interconnected neurons on the cloud.

6 Conclusions and Future Work

Cloud computing is becoming increasingly popular for multiple solutions in the IT world. Over the last couple of years, multiple cloud service providers such as Amazon, Google and Microsoft have entered the market with various solutions. Traditionally, this technology has been used to host user data and perform basic analysis. However, this paradigm is expected to change over the next few years. The cloud is expected to not only host data, but also perform computations that can be used for inference and prediction. As cloud technologies are improved, they may have an unexpected application in traditional scientific computation.

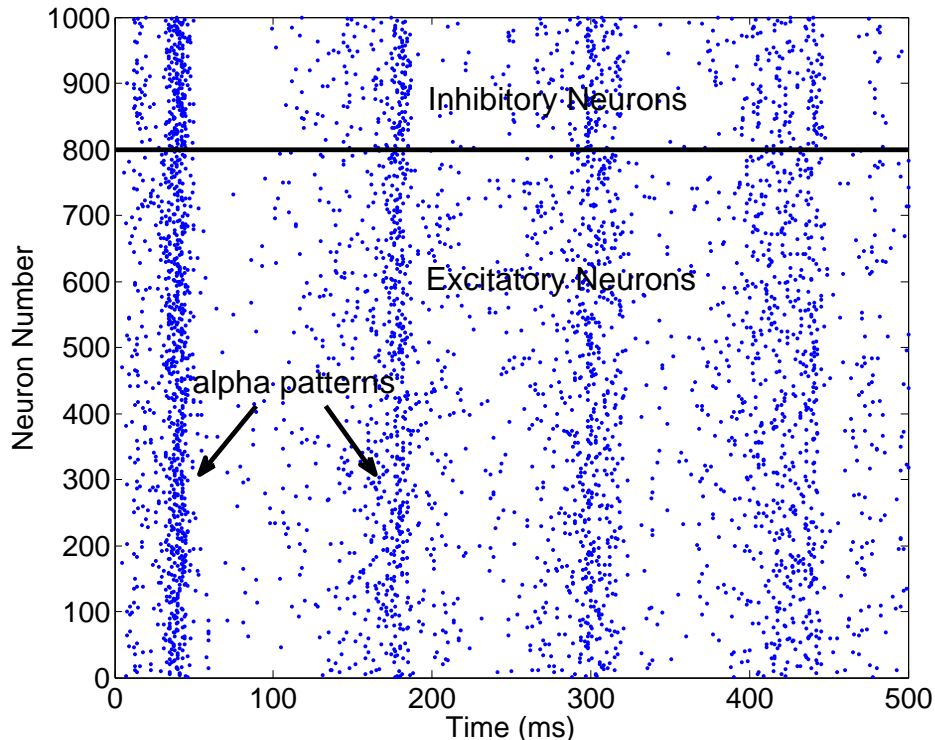


Figure 4: The firing in a network of 1000 neurons over a span of 500 ms. Computation performed using Hadoop.

In this work, we used the MapReduce paradigm to simulate a network of 1000 (800 excitatory and 200 inhibitory) neurons. We used the simple spiking model from [21] to model the dynamics of a single neuron. In the future, we intend to use Hodgkin-Huxley equations [20] to perform such computations. In particular, we use Hadoop (version 0.20.2) to perform the computations. The Map step is used to update the equations for individual neurons and the Reduce step is used to aggregate the results. As expected, we find that the results obtained from the MapReduce implementation are accurate.

Our primary computational overhead is the reading and writing to HDFS that needs to be performed at each step. Our current work is focused on replacing this step using graph computation solutions such as Giraph, Pregel and Oozie. We are also identifying other scientific computing problems that can be solved on the cloud. Additionally, Yarn, the next generation imple-

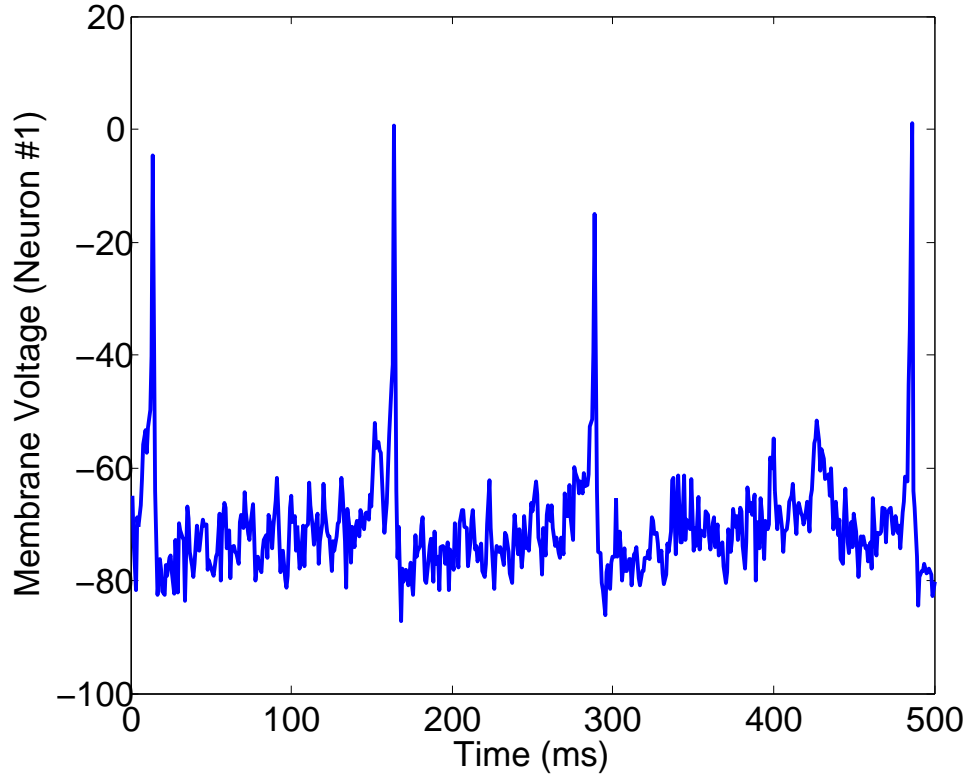


Figure 5: The membrane voltage v of neuron number 1 over a span of 500 ms. Computation performed using Hadoop.

mentation of MapReduce in Hadoop, is also expected to provide significant improvements in computational time. Thus, we expect the computational and financial cost of using the cloud to reduce over time. Not only will this make the cloud an increasingly attractive solution for scientific research, but also may provide cheap computing infrastructure to institutions in the third world.

References

- [1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

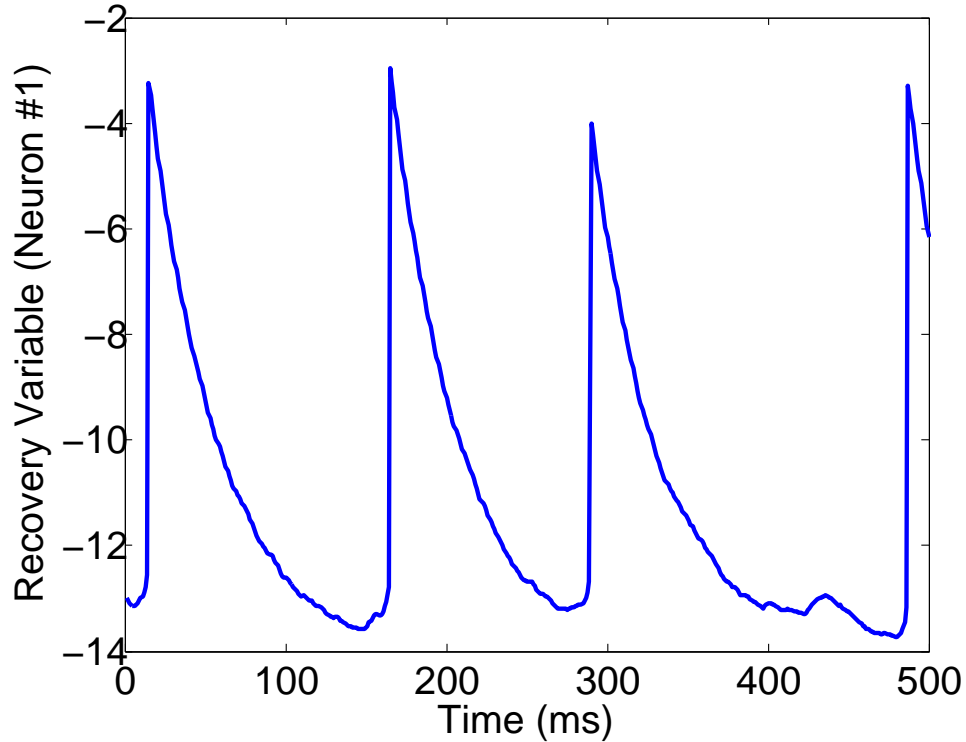


Figure 6: The recovery variable u of neuron number 1 over a span of 500 ms. Computation performed using Hadoop.

- [2] T. White. *Hadoop: The Definitive Guide*. O'Reilly Media, first edition, May 2009.
- [3] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 137–147, 2004.
- [4] Amazon. Amazon EC2 instance types. Retrieved from <http://aws.amazon.com/ec2/instance-types/>, 2010.
- [5] M. Djurfeldt, M. Lundqvist, C. Johansson, M. Rehn, Ö. Ekeberg, and A. Lansner. Brain-scale simulation of the neocortex on the ibm blue gene/l supercomputer. *IBM J. Res. Dev.*, 52(1/2):31–41, January 2008.
- [6] W. M. Washington, J. W. Weatherly, G. A. Meehl, A. J. Semtner, Jr., T. W. Bettge, A. P. Craig, W. G. Strand, Jr., J. Arblaster, V. B.

- Wayland, R. James, and Y. Zhang. Parallel climate model (PCM) control and transient simulations. *Climate Dynamics*, 16:755–774, 2000.
- [7] Bischof C. H., Green L. L., Haigler K. J., and Jr T. L. Knauff. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. Technical report, 1994.
- [8] W. Park, E. V. Belova, G. Y. Fu, X. Z. Tang, H. R. Strauss, and L. E. Sugiyama. Plasma simulation studies using multilevel physics models. *Physics of Plasmas*, 6(5):1796–1803, 1999.
- [9] R. K. Menon, G. P. Bhat, and M. C. Schatz. Rapid parallel genome indexing with mapreduce. In *Proceedings of the second international workshop on MapReduce and its applications*, MapReduce '11, pages 51–58, New York, NY, USA, 2011. ACM.
- [10] D. DeWitt and M. Stonebraker. Mapreduce: A major step backwards. Retrieved from <http://databasecolumn.vertica.com/database-innovation/MapReduce-a-major-step-backwards/>, August 2010.
- [11] J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Morgan and Claypool, first edition, 2010.
- [12] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.
- [13] Apache Foundation. *Apache Mahout: Scalable machine learning and data mining*.
- [14] E. U. Izhikevich. *Dynamical Systems in Neuroscience*. The MIT Press, first edition, 2006.
- [15] P. Faure and H. Korn. Is there chaos in the brain? I. Concepts of nonlinear dynamics and methods of investigation. *Comptes Rendus de l'Academie des Sciences - Series III - Sciences de la Vie*, 324(9):773 – 793, 2001.
- [16] S. S. Robertson, J. Guckenheimer, A. M. Masnick, and L. F. Bacher. The dynamics of infant visual foraging. *Developmental Science*, 7(2):194–200, 2004.
- [17] D. Hansel, G. Mato, and C. Meunier. Synchrony in excitatory neural networks. *Neural Computation*, 7(2):307–337, 1995.

- [18] M. Kawato, K. Furukawa, and R. Suzuki. A hierarchical neural-network model for control and learning of voluntary movement. *Biological Cybernetics*, 57:169–185, 1987.
- [19] Lopes da Silva, F. H., Hoeks, A., H. Smits, and L. H. Zetterberg. Model of brain rhythmic activity. *Biological Cybernetics*, 15:27–37, 1974. 10.1007/BF00270757.
- [20] A. L. Hodgkin and A. F. Huxley. A quantitative description of membrane current and its application to conduction and excitation in nerve. *Journal of Physiology*, 117(4):500–544, 1952.
- [21] E. M. Izhikevich. Simple model of spiking neurons. *IEEE Transactions on Neural Networks*, 14(6):1569–1572, 2003.
- [22] L. Ramakrishnan, K. Muriki, S. Canon, S. Cholia, J. Shalf, H.J. Wasserman, and N.J. Wright. Performance analysis of high performance computing applications on the amazon web services cloud. In *IEEE Second International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 159–168, 2010.